Aristotle: A Flexible Open-Source Software Toolkit for Semi-Automated Marking of Programming Assignments

Michael D. Adams
Dept. of Electrical and Computer Engineering, University of Victoria
Victoria, BC, V8W 2Y2, Canada
mdadams@ece.uvic.ca

***Abstract*—A new flexible open-source software toolkit to assist in the marking of programming assignments is presented. This toolkit automates the process of validating, building, and testing assignment submissions in addition to generating reports summarizing the results of building and testing student code. By using this toolkit (especially in conjunction with a tool like GitHub Classroom), the amount of effort required for assignment marking can be greatly reduced. The toolkit is programming-language neutral (i.e., does not mandate the use of a particular language) and provides a high degree of flexibility by allowing various operations to be customized via user-provided programs, which may be written in a scripting language of the user's choice.**

***Index Terms*—Programming assessment, software assessment tools, semi-automated grading, GitHub Classroom.**

I. Introduction

Many courses in engineering, computer science, and related disciplines require the marking of programming assignments. The marking of such assignments, however, can be extremely time consuming. Typically, an assignment submission must first be checked to ensure that it has been packaged correctly by the student. Then, all of the code must be built and tested. Moreover, to facilitate more thorough testing, one or more modified versions of the student code may sometimes need to be generated. For example, an instructor may wish to substitute their own test code in place of student test code. Often, a manual inspection of the source code is also required in order to consider criteria such as coding style, code structure, and documentation. A report that provides feedback to the student about the marking needs to be generated. Finally, the above process must be repeated for each student in the class, which may potentially be quite large. Since the marking process is potentially very time consuming, tools that can automate or otherwise assist with some parts of this assessment process are highly beneficial.

Over the years, many tools have been developed to assist with the marking of programming assignments [1]–[3], providing varying degrees of automation from semi- to fully-automated. Full automation of the marking process is often not feasible, since some evaluation criteria may be difficult for a computer algorithm to automatically assess in a meaningful way. For example, certain aspects of programming style are difficult to assess in an automated manner. Consequently, semi-automated schemes are often the most appropriate choice in many contexts. Of the many tools that have been developed, many impose significant restrictions on the user. For example, some tools are intended for use with only one particular programming language (often, Java) or one particular software build tool or one particular compiler tool chain, while other tools require the user to employ a specific scripting language for customization or defining test cases. Such restrictions often limit the utility of a tool.

In this paper, I present Aristotle, a new flexible open-source software toolkit to assist in the marking of programming assignments. This software provides a very general framework for semi-automated assignment marking. It is primarily intended for use in situations where the assessment of assignment submissions requires both testing the submitted code as well as performing a manual code inspection. Aristotle provides a means to automate most parts of the assessment process other than code inspection and final grade assignment. In particular, it automates: the importing of assignment submissions and performing basic validity checks on submissions, the building and testing of the code in the submissions, and the generation of a report summarizing the results of building and testing the code. Test case failures and other information included in the generated report can offer additional guidance for a manual code inspection. Aristotle can accommodate a wide variety of programming languages, compiler tool chains, and build tools, and allows for great flexibility in the types of assignments that can be handled. A large degree of flexibility is achieved by allowing the behavior associated with various operations to be specified by user-provided programs, which may use a scripting language of the user's choice (allowing the user to employ the language with which he/she is most comfortable). The current version of Aristotle is implemented primarily using Bash [4] scripts, consists of approximately 8900 lines of code, and should be reasonably portable across most modern Unix-based platforms. The Aristotle software can be obtained from its official Git repository [5] on GitHub.

II. Background

Before discussing the Aristotle software further, it is helpful to first introduce a very useful tool that can be employed in conjunction with Aristotle, namely, GitHub Classroom.

GitHub Classroom [6] is a popular free web-based tool for educators offered by GitHub (http://github.com). This tool allows instructors to easily create and manage a collection of Git repositories [7] for each programming assignment in a course. Students place their assignment submissions in Git repositories, which can then be retrieved by the instructor (or his/her delegate) after the submission deadline passes. GitHub Classroom provides a very convenient means for managing assignment submission. For this reason, as we will see later, Aristotle supports the use of GitHub Classroom as a mechanism for assignment submission.

The development of the Aristotle software grew out of a need for a better way to handle the marking of programming assignments for senior-undergraduate and graduate courses taught by the author, which employ the C++ and C programming languages in conjunction with build tools such as CMake [8] and Make [9]. In these courses, assignment submissions are assessed based on both testing and a manual code inspection. A software toolkit was desired that could be used to automate all but the manual code inspection and subsequent final mark assignment. Furthermore, a solution to this problem was sought that would avoid many of the shortcomings of other previously-developed tools and produce a toolkit that would be maximally useful to others. To achieve the desired objective, it was felt that the toolkit to be developed should: 1) accommodate a wide range of assignment types; 2) allow any programming language to be used for assignments; 3) allow an arbitrary tool to be used to build the code in assignment submissions (for languages requiring compilation), and provide direct support for the use of the build tools CMake [8] and Make [9]; 4) allow for a wide range of testing methodologies; 5) decouple assignment submission from other operations, and support a variety of mechanisms for assignment submission, including GitHub Classroom [6], Git repositories [7], Zip and Gzipped-Tar archives, and directories within the local filesystem; 6) provide a mechanism for creating modified versions of student code, which could be used, for example, to replace student test code with instructor test code; 7) allow the instructor to expose commonly-used build tools (e.g., CMake and Make) to students (if desired) by requiring them to provide build files (such as CMakeLists files or makefiles) as part of the assignment submission; 8) integrate well with GitHub Classroom; and 9) allow automation of the following actions: importing assignment submissions, performing basic validity checking of submissions, building and testing code, and generating a report summarizing the building/testing results (which can be used to provide additional context for a manual code inspection). Item 7 was considered important since, without direct exposure to build tools, students can successfully complete a course without actually knowing how to compile and link code [1].

III. Overview of Aristotle

With Aristotle, an **assignment submission** is simply a directory tree containing one or more files. The particular structure of the directory tree is arbitrary. For example, it may be flat (i.e., all files in the top-level directory) or have many levels of subdirectories. Normally, the top-level directory of the assignment submission is required to contain a file called `IDENTIFICATION.txt`. This file holds information about the student who authored the assignment submission (e.g., name, student ID, email address, and course section) and the assignment itself (e.g., assignment ID), and must follow a particular format in order to facilitate automatic extraction and validation of this data. (An example of an identification file can be found later in Listing 1.) Alternatively, a user may choose to encode the student/assignment identification information in a completely arbitrary manner as long as a program (which may be a script) is provided for extracting this information.

With Aristotle, each assignment is named by a unique ID and is defined as a collection of attribute-value pairs (i.e., a pair consisting of an attribute and its corresponding value). An **attribute** is simply some characteristic or property of an assignment. Some examples of attributes include: a short assignment description, a list of required files, a list of blacklisted (i.e., disallowed) files, and a list of packages defined for the assignment. All of the attribute-value pairs for a single assignment are placed in what is called an **assignment-definition file**. For each assignment, the user creates a corresponding assignment-definition file. Assignment definition files are then placed in a user-configurable directory called the **assignment-definition directory** so that these files can be located by Aristotle. Many assignment attributes have default values. Consequently, an assignment-definition file need only specify the value for a particular attribute if either it has no default value or the desired value differs from the default.

In order to process an assignment submission with Aristotle, the submission must be imported into what is called an **assignment workspace**. An assignment workspace is simply a directory tree with a particular layout known to Aristotle.

With Aristotle, the processing of an assignment submission normally consists of the following steps in order:

1) ***Import***. Create a new assignment workspace using an assignment submission from some input source, such as a Git repository or an archive file (e.g., a Zip or Gzipped-Tar archive).
2) ***Validate***. Perform some basic validity checks on the data imported from the assignment submission.
3) ***Generate packages***. Generate one or more variants of the files from the assignment submission, where each variant is known as a package. In the simplest case, only one package would be generated that is identical to the original student submission.
4) ***Configure***. Perform any pre-build configuration required for the packages generated in the previous step (i.e., step 3). What this configuration step does (if anything) depends on the particular build tool being used.
5) ***Build***. Build (e.g., compile and link) the code for the packages configured in the previous step (i.e., step 4).
6) ***Test***. Test the programs or libraries built in the previous step (i.e., step 5).

7) ***Generate report***. Generate a report summarizing the results of building and testing the packages produced in step 3.

The report generated in the last step above would then be used in conjunction with a manual code inspection in order to determine a final mark for the assignment submission.

With Aristotle, the instructor is free to choose the mechanism to be used for the collection of assignment submissions. This said, however, the use of GitHub Classroom is strongly recommended (especially for larger classes), as this allows much of the work associated with managing assignment submissions to be automated. If GitHub Classroom is employed, the source of the assignment-submission data for the import operation (in step 1 above) would be the Git repositories associated with GitHub Classroom, and all of the submissions for an entire class can be imported with a single command.

IV. Aristotle in More Detail

Now that the reader has been given a brief overview of Aristotle, we are ready to examine this software in more detail. To begin, we consider how assignments are defined. Then, we proceed to examine the various operations involved in the processing of an assignment submission.

A. Assignment Definitions

As mentioned previously, an assignment definition is a collection of attribute-value pairs, where each pair is used to specify a particular characteristic of an assignment. Each assignment definition is placed in a separate assignment-definition file in the assignment-definition directory. The assignment-definition file for the assignment with ID *id* must be named *id*`.asgn` (i.e., the assignment ID with the "`.asgn`" suffix added). Within the file, attribute-value pairs are specified one per line, with the attribute followed by its value (which may consist of multiple words). The hash character (i.e., "`#`") begins a comment that continues until the end of line. A backslash character (i.e., "`\`") at the end of a line causes the end of that line to be ignored, resulting in line continuation. Aristotle predefines a number of variables whose values can be used in assignment-definition files. The value of a variable *var* is accessed by the expression `${`*var*`}`. A fully-commented example of an assignment-definition file can be found later in Listing 2.

Each assignment has one or more packages, where each package corresponds to a transformed version of the original assignment submission. In turn, each package has a number of build targets and test targets. A build target names an artifact that can be generated by the software build process, such as an executable program or a library, and a test target names a test that can be performed on the software produced by the build process. Packages, build targets, and test targets have a number of attributes. For example, each of these items can be marked as either required or optional. The optional designation marks an item as being for instructor use only. The student usage of Aristotle only requests the processing of required (i.e., non-optional) items, while the instructor usage would request the processing of all (i.e., both required and optional) items. For example, test targets that are to be made available to the instructor only (not the student) would be marked as optional. A number of the attributes in an assignment definition are used to specify user-provided executable programs (which may be scripts) to be run to perform various tasks in the processing of an assignment, such as performing the test associated with a test target or applying a user-defined package-generation transformation (to be discussed later). Since no constraints are imposed on the scripting languages that can be employed for such programs, the user is free to employ whatever scripting language is most convenient. Generally, these programs are required to adhere to the convention that they should set their exit status to zero upon success and a non-zero value upon failure. Some key parameters to user-provided programs (such as the directories containing various files of interest) are made accessible through environment variables. Due to space constraints, it is not possible to describe herein all of the assignment properties and variables for assignment-definition files. Additional details can be found in the documentation available with the Aristotle software [5].

B. Importing Assignment Submissions

In order to process an assignment submission, the data for the submission must first be imported into an assignment workspace. This import operation is performed with the `ari_import` command, which creates a new assignment workspace and loads the data for an assignment submission into that workspace. The data for the assignment submission can be obtained from one of a variety of types of sources, including a Git repository, Zip archive, Gzipped-Tar archive, or a directory in the local filesystem. For convenience, multiple assignment submissions can be imported with a single invocation of the `ari_import` command (e.g., to import all of the assignment submissions from a class at once).

C. Validating Assignment Submissions

After an assignment submission has been imported into an assignment workspace, the imported data is typically checked to ensure that it meets certain very basic requirements. This process, known as validation, is performed by the `ari_validate` command. During validation, Aristotle automatically checks to ensure that, amongst other things: all required files/directories for the assignment are present; no disallowed (e.g., blacklisted) files/directories are present; and student identification information has been provided in the correct format. The user can also specify a program/script to perform additional validation checks beyond those performed automatically by Aristotle. Validation can be used to ensure that all assignment submissions conform to some prescribed organizational, formatting, or other requirements. This consistency in assignment submissions avoids many potential problems during marking. Students as well as the instructor may perform a validation operation. In this way, students can confirm that their assignment work passes some basic validity checks prior to submission. Moreover, an instructor might even

require that an assignment submission must successfully pass this validation test in order to be eligible for marking.

D. Generating Packages

For increased flexibility, instead of directly building and testing the assignment submission exactly in the form submitted by the student, Aristotle uses the submission to generate one or more transformed versions of the submission, called packages. Packages are generated using the `ari_generate` command. The default transformation for package generation is the identity, which simply produces a package that is identical to the original assignment submission. The user, however, may define their own custom transformations to be employed for package generation. Such transformations can be arbitrarily complex and are defined by having the user specify a program (e.g., script) that performs the transformation. In the simplest case, a user would simply create a single package for an assignment that uses the (default) identity transformation. In more complicated situations, however, user-defined transformations can be quite beneficial. For example, user-defined transformations might be used for such purposes as: 1) creating a variant of the assignment submission where student test code has been replaced by instructor test code; and 2) changing the organization of files in the assignment submission to facilitate easier processing with Aristotle.

E. Configuring and Building Packages

After having generated the packages associated with the assignment, the next step is to build the code in these packages. This is accomplished with the `ari_build` command. To allow increased flexibility, the building of the code is performed in two steps (in order): 1) configuring and 2) building. The configure operation performs any setup that is needed prior to the building of the software, while the build operation actually builds the software. What, in particular, each of these operations does (if anything at all) depends on the particular build tool selected by the user. In this regard, one of three options can be selected: 1) CMake, 2) Make, or 3) a user-specified tool. If the build tool is CMake, the configuration operation will invoke CMake to generate the build files for the native build tool employed on the platform being used, and the build operation will then invoke this native build tool with the build files produced during configuration. If the build tool is Make, the configuration operation does nothing (since no pre-build step is necessary), and the build operation invokes Make. If a user-defined build tool is selected, then the user must specify a program that can be invoked to perform each of the configure and build operations. If no build tool is selected, Aristotle will automatically choose between CMake and Make, depending on whether CMakeLists files or makefiles are provided in the package to be built. By default, all build targets for all packages for the assignment are built.

F. Testing Packages

After having built the code for the packages associated with the assignment, the code is ready to be tested. The testing of the software is accomplished with the `ari_test` command. Each package defines a number of test targets. Each test target corresponds to a test to be performed and is associated with a particular test program, which must be specified by the user. Test programs can perform any arbitrary tests, from very simple to very complex. In the interest of flexibility, Aristotle does not impose any particular testing methodology. A single test target could be used, for example, to test a single program (or library) or a collection of several programs and libraries. A test program might test a program by comparing actual program output against expected output, while a library might be tested using instructor test code or a test framework such as Google Test [10]. For each test target, a timeout can be specified. If the test does not complete before the timeout expires, the test is terminated and deemed to have failed. Such a timeout mechanism is helpful since incorrect code may become trapped in an infinite loop during testing. By default, all test targets for all packages for the assignment are tested. Depending on the preferences of the instructor, each test can either made visible to students or hidden from students.

G. Report Generation

After the packages for an assignment have been built and tested, a report can be generated (in PDF format) that summarizes the building/testing results. This is accomplished with the `ari_report` command. The report contains information such as: 1) the student who submitted the assignment; 2) a list of the files submitted; 3) a summary of the build and test results (e.g., whether or not each build target was successfully generated and whether or not each test target was successfully tested); 4) a log of the output generated by any failed build or test operations; and 5) listings of zero or more files from the original assignment submission. Item 4 can be used to diagnose the cause of any errors encountered during the building or testing of code, while item 5 is often convenient for the purpose of performing a manual code inspection. Since log files can occasionally become very large, log files that are unreasonably long are truncated before inclusion in the report in order to prevent reports of excessive size.

H. Optional Packages, Build Targets, and Test Targets

Typically, an instructor would not want students to have access to all of the tests used to assess the correctness of the code in an assignment submission. Preventing students from having access to test programs can be easily accomplished through the use of file permissions. That is, any test programs or test code to which students should not have access, can be placed in directories or files whose permissions are such that they are inaccessible by students. Any packages, build targets, and test targets associated with files to which the student does not have access can be tagged as optional (i.e., instructor only) so they can be skipped if encountered when a student invokes an Aristotle command (in order to avoid file permission errors).

I. Other Operations

Aristotle provides quite a number of other commands in addition to those mentioned so far. For example, the `ari_precheck` and `ari_process` commands are each provided as a convenience, replacing common sequences of operations with a single command. The `ari_precheck` command takes one or more assignment submissions as input. It imports and validates the assignment submissions, generates any non-optional packages, builds and tests any non-optional targets, and generates a report. This command is intended for the student to use as a last sanity check before submitting their assignment work. The `ari_process` command is similar to `ari_precheck`, except that it takes assignment workspaces as input (instead of assignment submissions) and considers all packages, build targets, and test targets, including ones marked as optional (i.e., instructor only). This command is intended for use by the instructor. Aristotle also provides several commands for interfacing with GitHub Classroom, such as commands for: login and logout (e.g., creating and deleting access tokens), creating and deleting repositories, listing the repositories for a given assignment, tagging repositories, copying repositories, and extracting a particular snapshot from a repository.

J. Other Remarks

With Aristotle, the execution of commands is performed with the privileges of the invoking user. Consequently, for reasons of security, when the instructor invokes commands that may run student programs, this should be done under an account with limited privileges. This is needed to safeguard against deliberate or accidental destructive actions taken by student programs.

V. Example

For illustrative purposes, we will now consider a simple example of an assignment for which Aristotle is to be used to assist in marking. In this assignment, the student is required to write a C++ program that reads whitespace-delimited words from standard input and writes to standard output whether each word is a palindrome (i.e., the program tests for palindromes). CMake is to be used as the build tool, and the student must provide a CMakeLists file for CMake that defines an executable target named `is_palindrome`. The assignment is to be named by the assignment ID `palindrome`. The assignment submission has the following three required files, all of which are located in the top-level directory of the submission: 1) `IDENTIFICATION.txt`, a file which contains student and assignment information; 2) `CMakeLists.txt`, the build configuration file for CMake; and 3) `is_palindrome.cpp`, the single C++ source code file for the `is_palindrome` program. An example of an `IDENTIFICATION.txt` file that might be used by a student in assignment submission is given in Listing 1. Example listings of the other two required files for the assignment submission are not shown, since they are not critical to the discussion herein, but can be obtained from the Aristotle software distribution [5]. An

Listing 1. Identification file (`IDENTIFICATION.txt`).

```
1 name "Jane Doe"
2 student_id "V12345678"
3 email "jdoe@uvic.ca"
4 section "T01"
5 assignment "palindrome"
```

assignment-definition file called `palindrome.asgn` has been created in the assignment-definition directory with the contents shown in Listing 2. From this listing, we can see that the assignment defines a single package called `original`. This package has one build target called `is_palindrome` and one test target called `is_palindrome`. The test target `is_palindrome` is used to test the correct behavior of the `is_palindrome` program. The actual test is specified as being performed by a program (which is a Bash script) called `palindrome-is_palindrome-test`. This program is located in the directory `../private/bin`, where this path name is interpreted relative to `${ARI_ASSIGNMENTS_DIR}`, which is the assignment-definition directory. When invoked by Aristotle, this program is passed the command-line argument `${ARI_DERIVED_DIR}`, which is the name of the directory containing the built code (i.e., the `is_palindrome` program to be tested). The test program simply runs the `is_palindrome` program with several inputs and tests if the program generates the correct output in each case. For additional information about the assignment-definition file, the reader is referred to the detailed comments in the listing.

Suppose now that GitHub Classroom is being used for assignment submission. In particular, suppose that all of the assignment submissions are in Git repositories managed by GitHub Classroom under the GitHub organization `$org` and are associated with the assignment name `palindrome`. Then, all of these submissions can be imported into Aristotle assignment workspaces under the directory `workspaces` and fully processed to yield reports (in PDF format) with the short command sequence given in Listing 3. For each submission, an assignment workspace will be created under the directory `workspaces`, and each workspace will contain the generated report in a file named `report.pdf`. In the command sequence, the `ari_gc_lsrepo` command is used to obtain the URLs for all of the Git repositories managed by GitHub Classroom under the organization `$org` for the assignment `palindrome`.

VI. Conclusions

In this paper, I have presented a new flexible open-source software toolkit for the semi-automated assessment of programming assignments. This toolkit can be used with any programming language, build tools, and compiler tool chains. By allowing the user to customize various key operations in the processing of assignment submissions, considerable flexibility is achieved. The Aristotle software can be downloaded from GitHub [5]. By using software like Aristotle, the amount

Listing 2. Assignment definition file (`palindrome.asgn`).

```
# Specify a name or short description for the assignment.
name "Palindrome Test"
# Specify the files that must be included in the assignment submission.
required_files IDENTIFICATION.txt CMakeLists.txt is_palindrome.cpp
# Specify the files from the original submission that are to be included in
# the generated report.
report_files CMakeLists.txt is_palindrome.cpp
# Specify the packages defined for this assignment (i.e., one called
# "original").
packages original

# Specify a name or short description for the "original" package.
package-original/name "The original code exactly as submitted by the student."
# Specify the build targets for the "original" package (i.e., one called
# "is_palindrome").
package-original/builds is_palindrome
# Specify the test targets for the "original" package (i.e., one called
# "is_palindrome").
package-original/tests is_palindrome

# Specify a name or short description for the "is_palindrome" build target.
package-original/build-is_palindrome/name "Build the is_palindrome program."

# Specify a name or short description for the "is_palindrome" test target.
package-original/test-is_palindrome/name "Test the is_palindrome program."
# Request that the test should only be performed if the associated build
# target "is_palindrome" is successfully built.
package-original/test-is_palindrome/depends_on build-is_palindrome
# Set the maximum time (in seconds) allowed for the test to 10.
package-original/test-is_palindrome/timeout 10
# Specify the command (including arguments) to be invoked to perform the test.
# Note: ${ARI_ASSIGNMENTS_DIR} is the directory containing this assignment
# definition file and ${ARI_DERIVED_DIR} is the directory containing the
# program to be tested.
package-original/test-is_palindrome/test \
  ${ARI_ASSIGNMENTS_DIR}/../private/bin/palindrome-is_palindrome-test \
  ${ARI_DERIVED_DIR}
```

Listing 3. Command sequence

```
ari_import -p workspaces/ $(ari_gc_lsrepo -o $org -f ssh -a palindrome)
ari_process workspaces/*
```

of work required to mark programming assignments can be greatly reduced.

REFERENCES

[1] K. M. Ala-Mutka, "A survey of automated assessment approaches for programming assignments," *Computer Science Education*, vol. 15, no. 2, pp. 83–102, Jun. 2005.

[2] C. Douce, D. Livingstone, and J. Orwell, "Automatic test-based assessment of programming: A review," *ACM Journal of Educational Resources in Computing*, vol. 5, no. 3, pp. 1–13, Sep. 2005.

[3] R. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppala, "Review of recent systems for automatic assessment of programming assignments," in *Proc. of 10th Koli Calling International Conference on Computing Education Research*, Oct. 2010, pp. 86–93.

[4] "Bash," 2017, https://www.gnu.org/software/bash.

[5] "Aristotle GitHub page," 2017, https://github.com/mdadams/aristotle.

[6] "GitHub Classroom," 2017, https://classroom.github.com.

[7] S. Chacon and B. Straub, *Pro Git*, 2nd ed. Apress, Nov. 2014, https://git-scm.com/book/en/v2.

[8] K. Martin and B. Hoffman, *Mastering CMake — A Cross-Platform Build System — CMake 3.1*. Kitware, Jan. 2015.

[9] S. I. Feldman, "Make — a program for maintaining computer programs," *Software: Practice and Experience*, vol. 9, no. 4, pp. 255–265, Apr. 1979.

[10] "Google Test," 2017, https://github.com/google/googletest.